

Arduino

Advanced Projects

Created as a companion manual to the Toronto Public Library Arduino Kits.



Arduino Advanced Projects

Copyright © 2017 Toronto Public Library. All rights reserved.

Published by the Toronto Public Library.

Table of Contents

TABLE OF CONTENTS	2
PREFACE	3
ARDUINO UNO TOUR.....	5
GETTING TO KNOW YOUR BREADBOARD.....	6
SAFETY TIPS.....	ERROR! BOOKMARK NOT DEFINED.
INTRODUCTION.....	10
CONTROLLING A SERVO WITH A POTENTIOMETER.....	16
CREATING A 30 SECOND COUNTDOWN TIMER.....	20
WHAT IS AN H-BRIDGE?	26
H-BRIDGE CONTROLLED BY AN ARDUINO	28
H-BRIDGE USING POTENTIOMETER.....	31
RECOMMENDED RESOURCES	37

Preface

Thank you for borrowing Toronto Public Library's Arduino Kit. Please return this kit to the Digital Innovation Hub from which it was borrowed.



Borrowing Arduino Kits

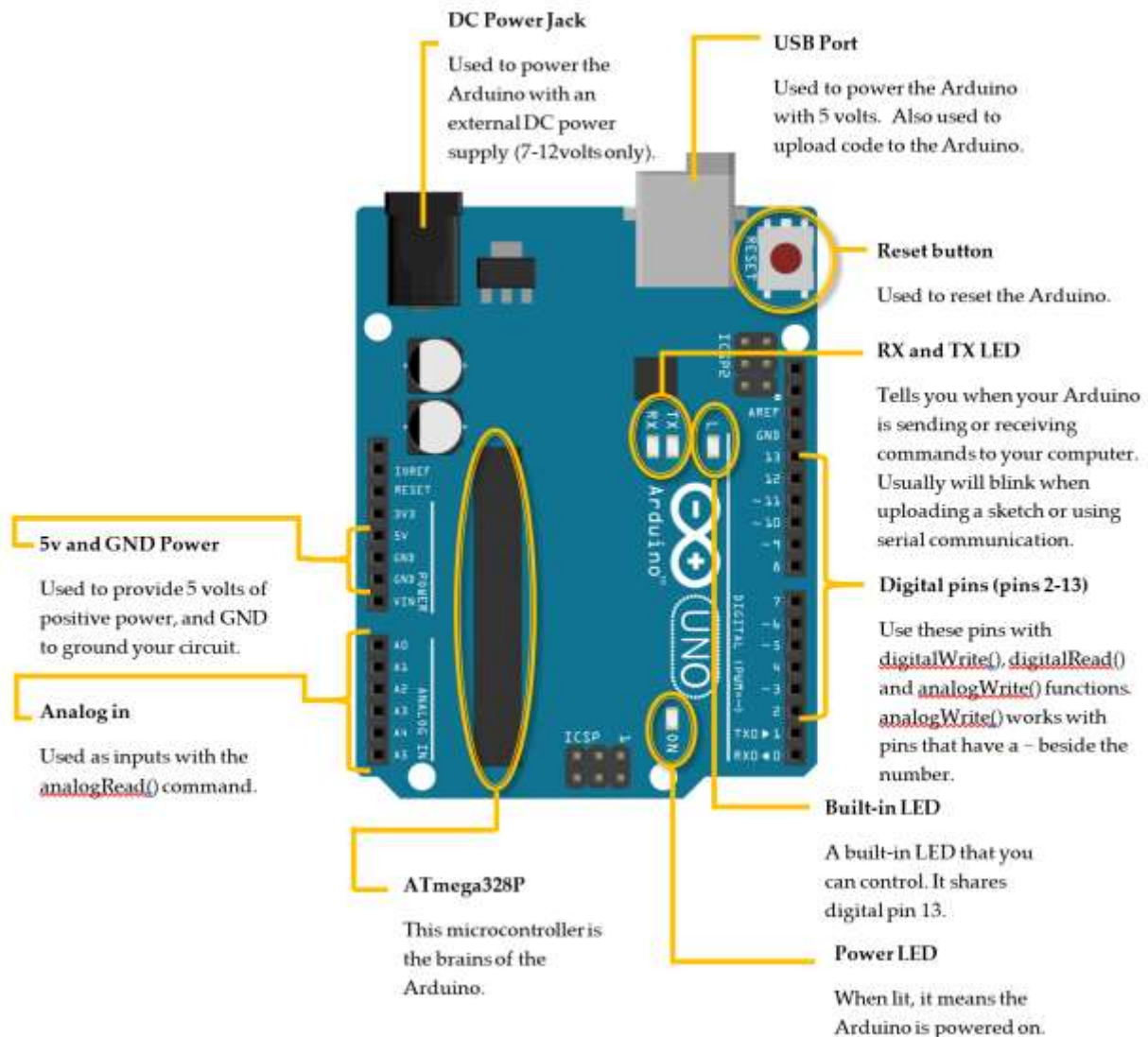
- Arduino Kits are available to Toronto Public Library customers with a valid Teen (13-17), Adult – Under 25 (18 – 24), or Adult (25+) library card.
- Holds cannot be placed on the Arduino Kits.
- You can only borrow one Arduino Kit at a time. Each kit can be borrowed for 21 days and cannot be renewed.

Fines Per Day and Maximum Fines for Arduino Kits		
CARD TYPE	FINE AMOUNT PER DAY	MAXIMUM YOU WILL BE CHARGED FOR EACH LOAN PERIOD
Adult	\$0.35	\$14.00
Adult Under 25 (18-24)	\$0.15	\$6.00
Teen (13-17)	\$0.15	\$6.00

- If you lose an Arduino Kit, you will be charged the purchase price of the Arduino (\$50). The library does not accept a replacement Arduino or an item of equal value.
- If the Arduino Kit is overdue by more than 40 days, the library considers it lost. If you find the kit within 6 months of paying the replacement cost you can get a refund, minus any overdue fines so please keep your receipt.
- Please report damaged equipment or missing parts to the Digital Innovation Hub staff from which it was borrowed. Damaged Arduino boards and kits are subject to replacement purchase fees.

Arduino Uno Tour

Time Required: 10 minutes



Spend a few moments looking at the diagram below and compare it to the Arduino included in your kit.

The Arduino has been labeled to help you learn all the different connectors and parts.

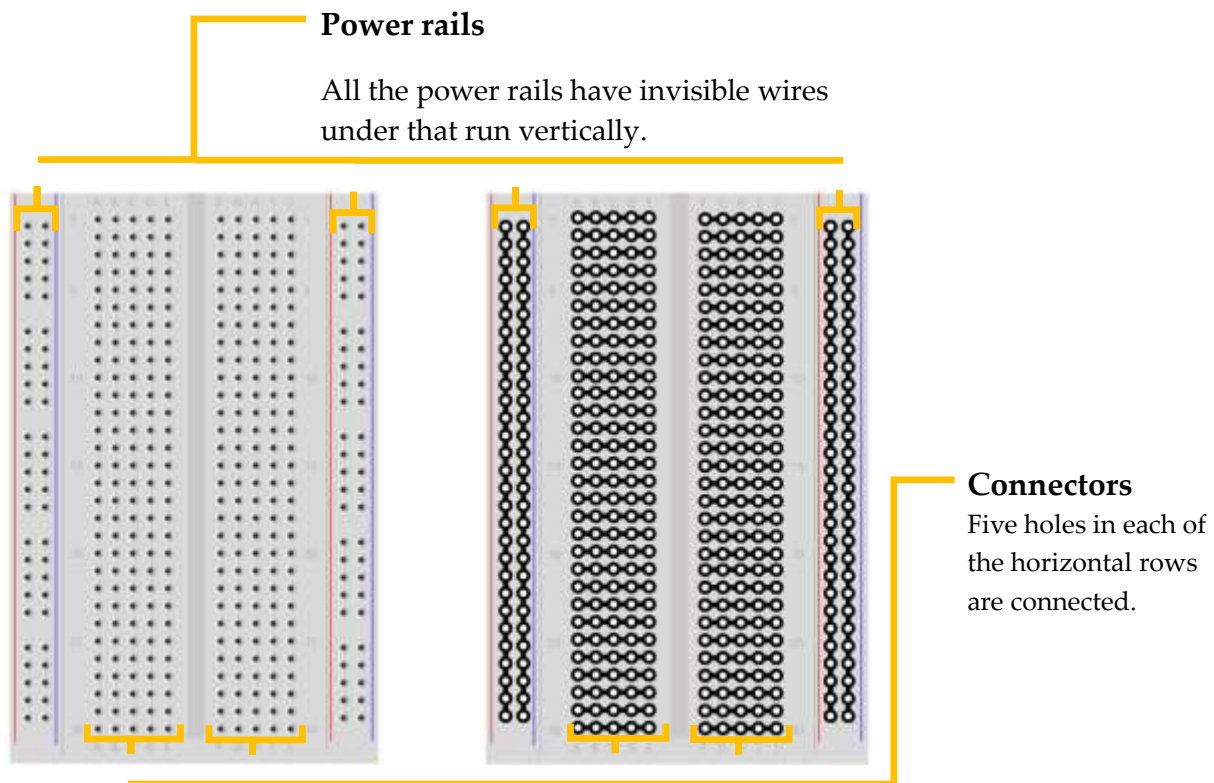
Getting to Know your Breadboard

Time Required: 20 minutes

Video resources about breadboards: <http://goo.gl/6HPHbg>

In order for us to connect our tiny components together, we need our breadboard. A breadboard is great for prototyping since it does not create a permanent connection between components like soldering does. Everything is held together by friction when you insert them into those tiny holes inside your breadboard.

Remember: If you have any questions, or need some extra help, feel free to visit a Digital Innovation Hub at the Toronto Public Library for classes or assistance.



What a breadboard looks like if we could see the wires under the breadboard

The previous page is an example of how a breadboard usually looks; on the right is how a breadboard would look if we could see the wires that connect all of the holes together. Those hidden wires are used to connect all your components to each other while you prototype.

Take a look at your breadboard. You may have noticed that the breadboard holes are all labeled A to J (vertically) and from 1-30 (horizontally). This is used to indicate where to place your components.

Throughout this guide, we will be asking you to place your components in very specific holes within your breadboard. For example, we might ask you to put a wire into hole **3E**. Now is the time to get familiar with the layout of your breadboard.

Safety Tips

Toronto Public Library's Arduino Kits use low voltage electricity and are not inherently dangerous. However, safety is always important when working with electrical circuits. Please follow the safety tips and instructions in this manual at all times.

Expert Tip: Always treat electronic projects as if they could have potentially dangerous voltages.

Each project has been planned and mapped out for you. Please take the time to read and thoroughly review the project instructions from beginning to end before you begin. Ensure that wires are connected accurately and in accordance with the diagrams provided. Not following the instructions as specified may result in personal injury or damage to the equipment.

Expert Tip: Turn off all power sources before modifying the circuit. Keep your Arduino unplugged while you are connecting wires and parts. Only connect it to the computer after your setup matches the diagram provided.

Keep your work surface clear when using this kit and maintain an orderly and safe work environment. Keep food and drinks away from the work area while working with your Arduino kit. Always unplug the Arduino when not in use. After using the kit, return all the parts to their proper storage place.

Expert Tip: Place the Arduino on a non-metal surface and refrain from working on metallic surfaces.

Warnings

- This kit is not a toy and is not appropriate for small children. Small parts may present a choking hazard. Not for children under 3.
- Avoid touching the exposed end of ground and power wires when connected to the Arduino.
- Use only the materials provided in the Arduino Kit.
- Do not make alterations or perform major repairs on the Arduino Kit.
- No soldering with the TPL Arduino kit.
- Do not use lithium ion batteries, they may explode when shorted
- Do not use on metallic surfaces, such as your Macbook. Place the Arduino on a non-metal surface and refrain from working on the surface of your Macbook.
- The library is not responsible for damage to any equipment and hardware used with the kit, including personal computers, laptops or tablets.
- Unplug the Arduino when not in use.
- Turn off/disconnect all power sources before modifying a circuit. While you're connecting components, keep your Arduino unplugged. Only connect it to a computer or power source after the circuit is complete.

Introduction

An Arduino is a microcontroller; a small, simple computer. It is designed specifically for beginners who are new to coding and electronics. You can learn more about the Arduino at <https://www.arduino.cc/en/Guide/Introduction>.

There are thousands of projects you can build with an Arduino.

Parts in This Kit

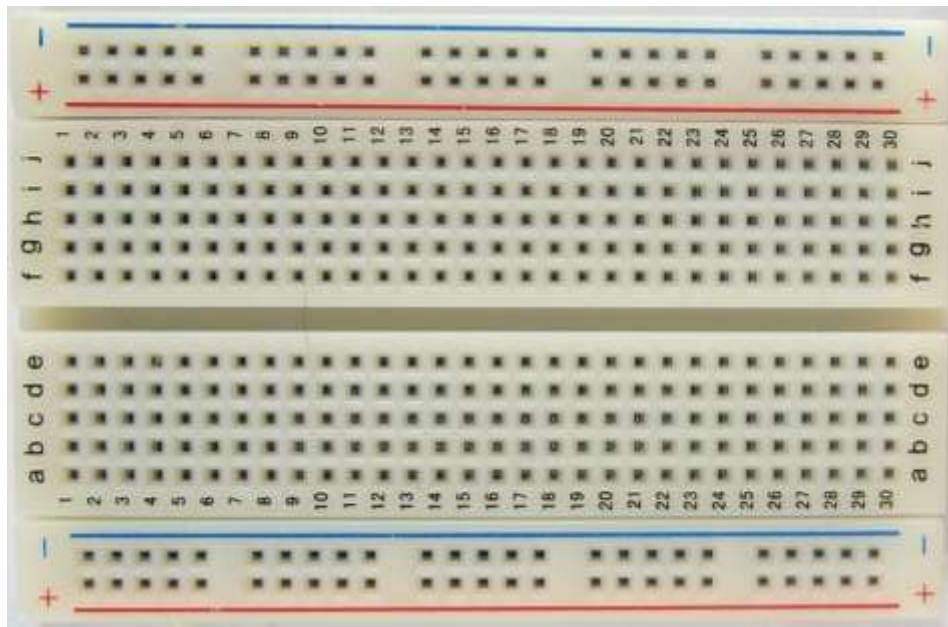
Your assembled kit includes all the parts you'll need for the Arduino projects outlined in this manual. When you're done with your projects, please return the parts to their proper slots, as indicated in this diagram, for the next person to enjoy.



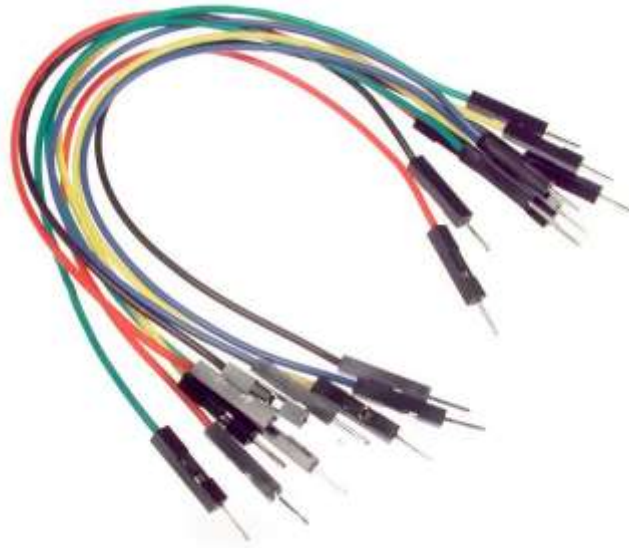
There are different types of Arduinos. This kit uses a blue **Arduino Uno** board. The different parts on the Arduino are labelled in white.



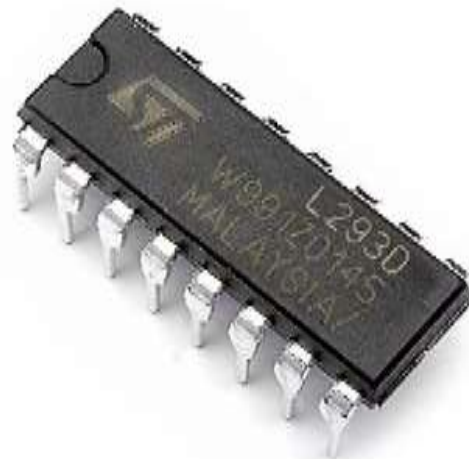
The **USB cable** is used to connect the Arduino to your computer.



The **breadboard** lets you build circuits. It has a series of holes where you can insert wires to create circuits. The magic of a breadboard is that it's reusable, and you don't need to solder (permanently joining components together to form a circuit by melting metals).



Jumper wires are used to create electric circuits and can be inserted into the breadboard.



A **H-Bridge IC Chip (model L293NE)** is an electronic integrated circuit chip that allows a voltage to be applied across a load (like a motor) in opposite directions. These are used in robotics so that you can control the direction of two motors independently.



DC Motor uses electricity to convert it to rotational mechanical energy. Connect the positive and negative to power and it will spin. Reverse the positive and negative pin to reverse the power flow and it will spin the opposite direction.

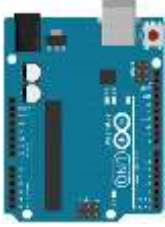
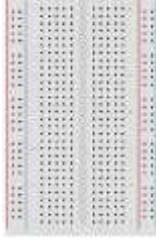








Potentiometers are a manually adjustable variable resistor with 3 terminals. Two terminals are connected to both ends of a resistive element, and the third middle terminal connects to a sliding contact, called a wiper, moving over the resistive element. The position of the wiper determines the output voltage of the potentiometer.



Servo Motor can be commanded to rotate to a specific angle. These motors cannot rotate continuously and only have a movement of 180 degrees.

Part Inventory for the Advanced Kit

 <p>1x Arduino</p> <p><i>The Arduino is the microcontroller and brains of our project. It stores programs and processes inputs and outputs.</i></p>	 <p>1x Breadboard</p> <p><i>The breadboard is used to temporarily connect multiple components and wires together during prototyping.</i></p>	 <p>16x Jumper Wires</p> <p><i>Jumper wires connect the components completing a circuit. Note: The colour of the wires do not matter when building the projects.</i></p>	 <p>1x Pushbutton</p> <p><i>When you push the button, it completes the circuit.</i></p>
 <p>1x H-Bridge IC chip</p> <p><i>Used to control the direction of two motors.</i></p>	 <p>1x Potentiometer</p> <p><i>Can be used to detect the position of a knob when connected to the Arduino analog input pins.</i></p>	 <p>1x DC Motor</p> <p><i>It is your standard DC motor. It spins when you apply power.</i></p>	 <p>1x Servo Motor</p> <p><i>Servo motor can be commanded to rotate to a specific angle (0-~180 deg).</i></p>

Controlling a Servo with a Potentiometer

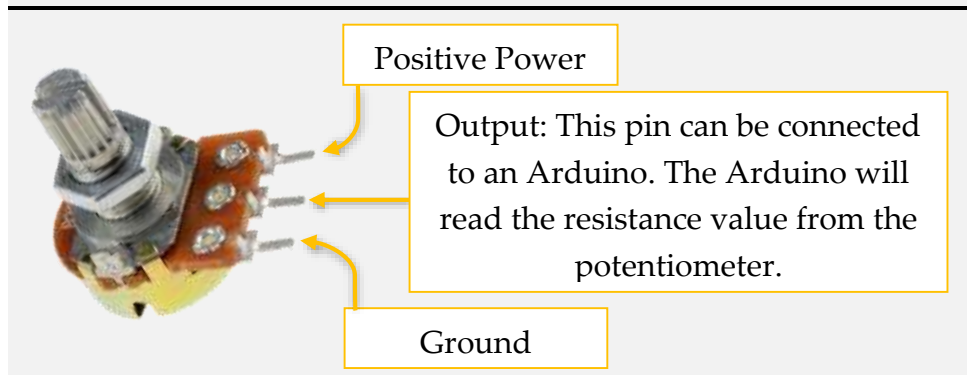
Time Required: 25 minutes

In this project, we will be using a potentiometer to control the movement of our servo motor. When we twist our potentiometer knob, the servo will also rotate approximately the same amount. Remember, it doesn't matter what color the wires are in the diagram compared to the ones you use.



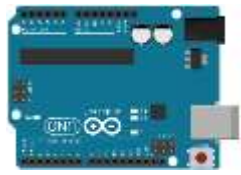




Potentiometers are like a dynamic resistor, as you twist, the resistance value changes (this will make the voltage change from 0-5v). These changes in resistance is what our Arduino will read and convert into a number between 0-1023.

What are the pins on the included Potentiometer?

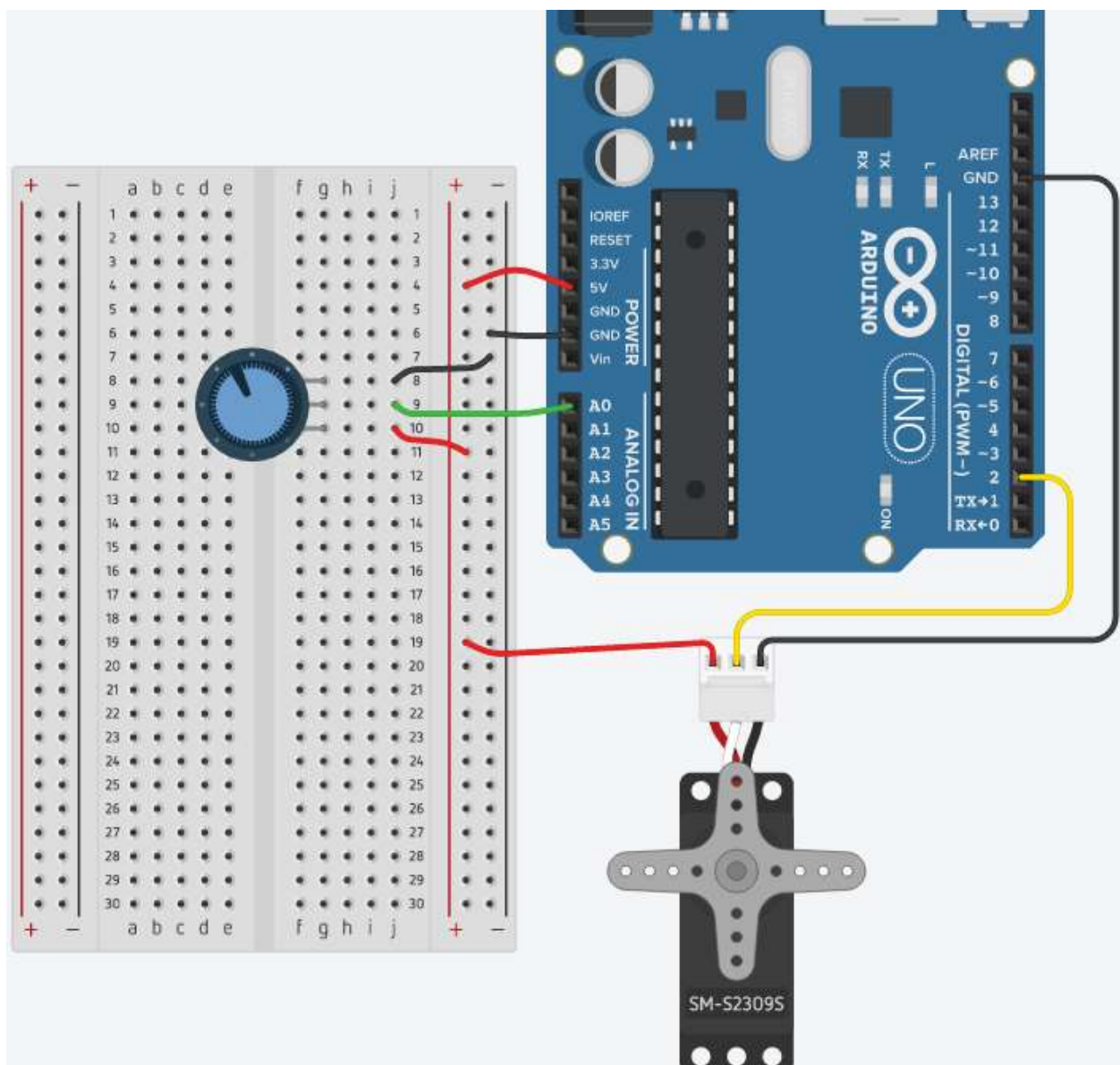


A servo motor, is a special type of motor that can be used to rotate precisely. There is a sensor inside the motor that keeps track of how much it rotates. This allows us to tell the motor to rotate by degrees and it will stop at the correct location.

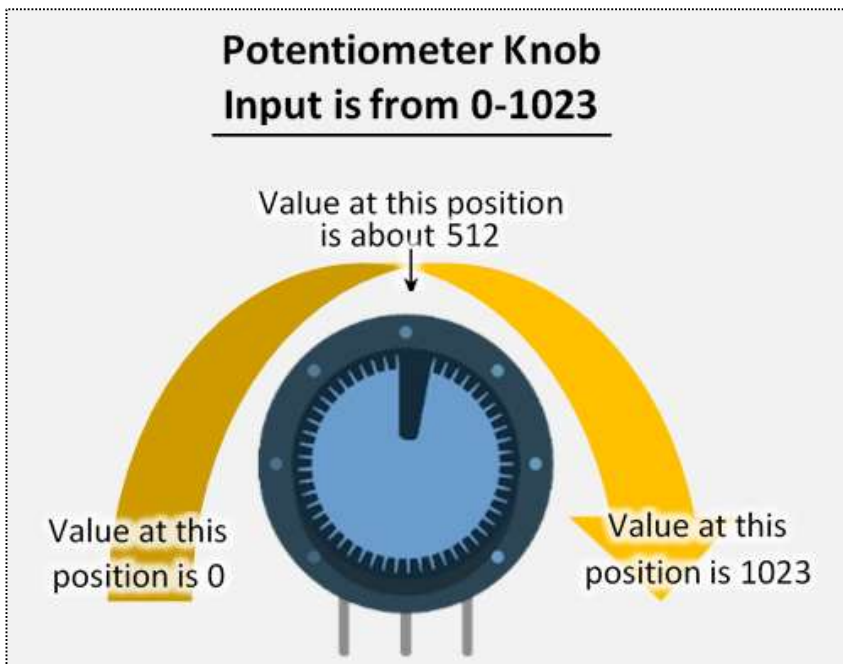
Required Components

				
1x Arduino	1x Breadboard	3x Jumper wires	1x potentiometer	1x Servo Motor

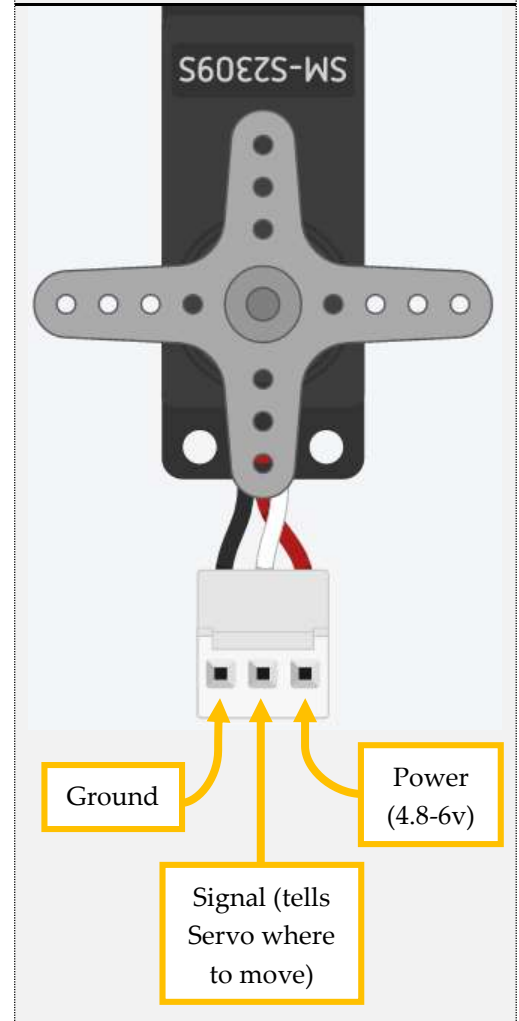
Wiring



Diagrams for this project



What are the pins on our Servo Motor?



Code

Line 1.
Line 2.
Line 3.
Line 4.
Line 5.
Line 6.
Line 7.
Line 8.
Line 9.
Line 10.
Line 11.
Line 12.
Line 13.
Line 14.
Line 15.
Line 16.
Line 17.
Line 18.
Line 19.
Line 20.
Line 21.
Line 22.
Line 23.
Line 24.
Line 25.
Line 26.

```
#include <Servo.h>
```

Include the ability to use the Servo motor library (enables us to use the servo commands).

```
const int servoPin = 2;
```

```
const int userInputPin = A0;
```

Create two constant integer variables. Variable *servoPin* will store the number 2, and *userInputPin* will store the number A0.

```
Servo myservo;
```

Create a new servo called *myservo*. We give it a name since we could control more than one servo at a time.

```
int pos = 0;
```

Create an integer variable with the value of 0. This will be used to store the position the servo should move to.

```
void setup() {
```

```
    myservo.attach(servoPin);
```

Tell the Arduino to use (attach) the servoPin (pin #2) to the servo called "*myservo*".

```
    pinMode(userInputPin, INPUT);
```

Set the potentiometer that is connected to *userInputPin* (pin #A0) as an INPUT.

```
}
```

```
void loop() {
```

```
    pos = analogRead(userInputPin);
```

Analog read the inputPin (A0), and store the value in our variable called *pos*.

```
    pos = map(pos, 0, 1023, 0, 180);
```

Map the raw sensor data (our sensor gives us a value between 0-1023) to a value between (0-255) and store the converted value back to the variable *pos*.

```
    myservo.write(pos);
```

Move the servo motor (called *myservo*) with the calculation we did previously stored in the variable *servoPos*.

```
    delay(100);
```

Delay (pause) for 100 microseconds. This will allow the servo motor to go to its position before being told to move again.

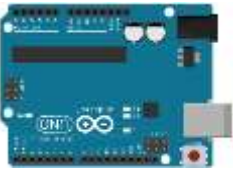
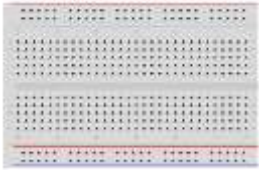



```
}
```

Creating a 30 Second Countdown Timer

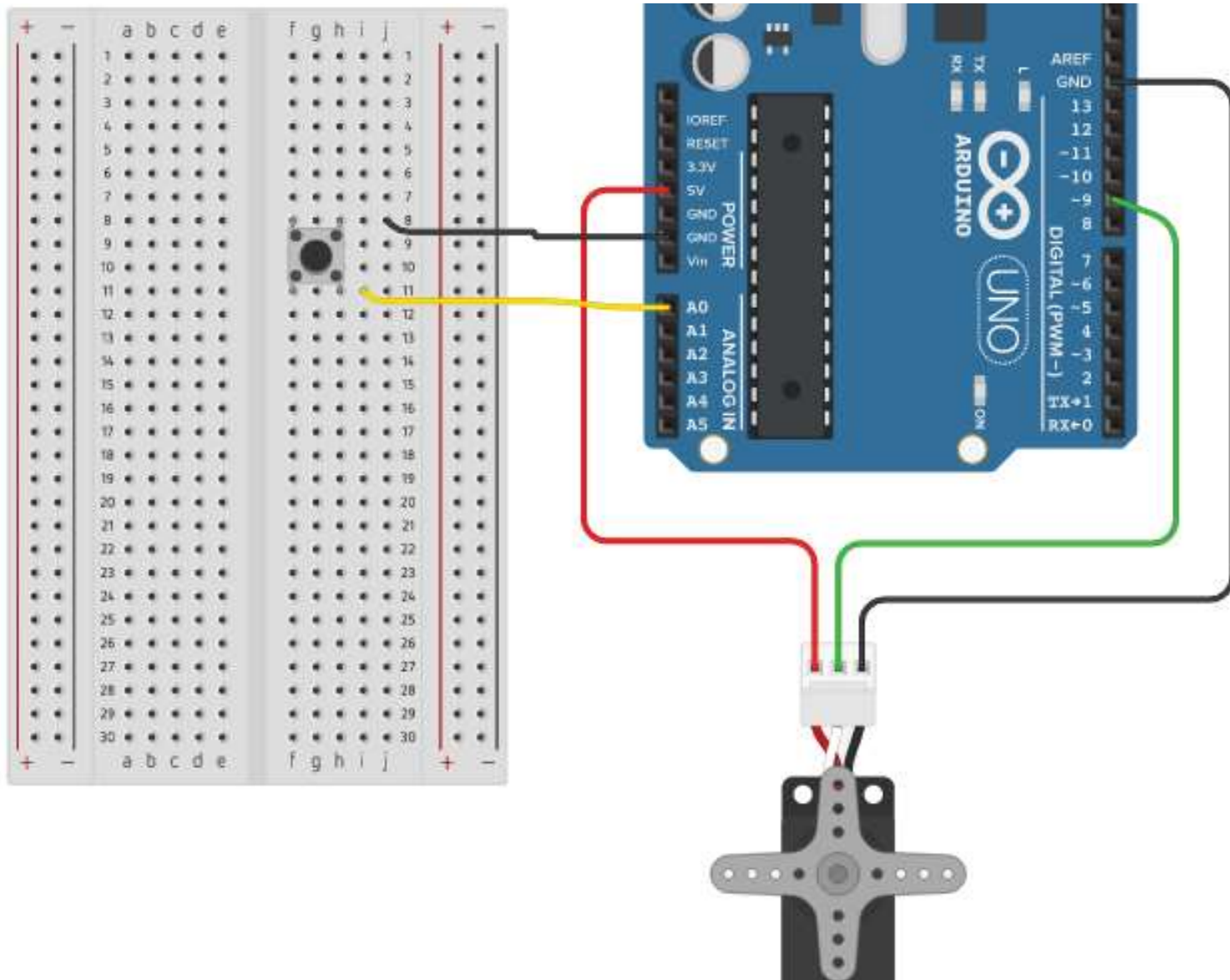
Time Required: 40 minutes

In this project, we will be using a servo motor as a countdown timer. The Servo motor will rotate 3 degrees of movement for every second for a total of 30 seconds. We have a button that is connected to the Arduino so we can restart the 30 second timer when needed.

Required Components

				
1x Arduino	1x Breadboard	5x Jumper Wires	1x Pushbutton	1x Servo Motor

Wiring



Code

Line 1. `#include <Servo.h>`

Include the ability to use the Servo motor library (enables us to use the servo commands).

Line 2. `Servo myservo;`

Create a new instance of a servo called *myservo*. We give it a name since we could control more than one servo at a time.

Line 3. `const int servoPin = 9;`

Create a constant integer variable called *servoPin* with a value of 9 (the pin the servo is using to be controlled).

Line 4. `const int buttonPin = 2;`

Create a constant integer variable called *buttonPin* with a value of 2 (the pin the button is connected to).

Line 5. `int servoPos = 0;`

Create an integer variable called *servoPos* with a value of 0. This is used to store the position the servo should go.

Line 6. `int timerCurValue = 0;`

Create an integer variable called *timerCurValue* with a value of 0. This is used for how much time has elapsed.

Line 7. `const int timerSetValue = 30;`

Create a constant integer variable called *timerSetValue* with a value of 30. This is used for how long we want the timer for.

Line 8. `void setup() {`

Line 9. `myservo.attach(servoPin);`

Line 10. `Serial.begin(9600);`

Line 11. `myservo.write(servoPos);`

Line 12. `pinMode(buttonPin, INPUT_PULLUP);`

Line 13. `delay(2000);`

Line 14. `}`

Tell the Arduino to use (attach) the servoPin (pin #9) to the servo called "myservo".

Setup serial monitor at a baud rate (speed) of 9600.

Set our servo (myservo) when the arduino turns on or restarts to the position at 0 degrees from the variable servoPos.

Set out button that is connected to buttonPin (pin #2) as an INPUT_PULLUP.

Wait two seconds before running the rest of our program. This will give our servo a moment to move to its 0 degree position if it needed to move.



What is INPUT_PULLUP?

*When setting an Arduino Uno pin as **INPUT_PULLUP** this activates the internal 20kohm resistor in the processor. This allows us to connect a button to ground, then to the pin we want the button to be connected to on the Arduino without using a resistor. This however will reverse the logical of reading the button compared to using **INPUT**. When the button is not pressed the pin will read **HIGH**, and when the button is pressed it will read **LOW**.*

```
Line 15. void loop() {
```

```
Line 16.   if(digitalRead(buttonPin) == LOW) {
```

```
Line 17.     timerCurValue = 0;
```

```
Line 18.     Serial.println("*** TIMER RESTARTED ***");
```

```
Line 19.   }
```

```
Line 20.   servoPos = timerCurValue * 3;
```

```
Line 21.   Serial.print("Timer: "); Serial.println(timerCurValue);
```

```
Line 22.   Serial.print("Position: "); Serial.println(servoPos);
```

Read if the button is pressed. Since the buttonPin (*pin #2*) is using *INPUT_PULLUP* instead of *INPUT*, the button pin will read *LOW* when pressed. When it is pressed reset the timer back to 0 and output the text **** TIMER RESTARTED **** in the serial monitor.

The timer starts at 0. Use the current elapsed time, multiple it by 3 and store this calculation in the variable *servoPos*. This means for every second the servo will move 3 degrees.

Output through the serial monitor the current elapsed timer from the variable *timerCurValue* and output the current servo position from the variable *servoPos*.


```
Line 23. Serial.println();
```

Output a line break to our serial monitor. This is only to make it easier to read the outputs and is for aesthetics.

```
Line 24. myservo.write(servoPos);
```

Move the servo motor (called *myservo*) with the calculation we did previously stored in the variable *servoPos*.

```
Line 25. if (timerCurValue < timerSetValue) {
```

```
Line 26.     timerCurValue = timerCurValue + 1;
```

If the current time (using our variable *timerCurValue*) is less than the set timer time (using the variable *timerSetValue* which is 30 seconds), increase the current time by one.

```
Line 27. }
```

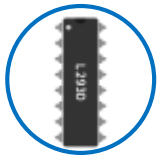
```
Line 28. delay(1000);
```

Wait one second before looping again. This makes it take one second to increase the *timerCurValue* variable by one and move the servo 3 degrees every second.

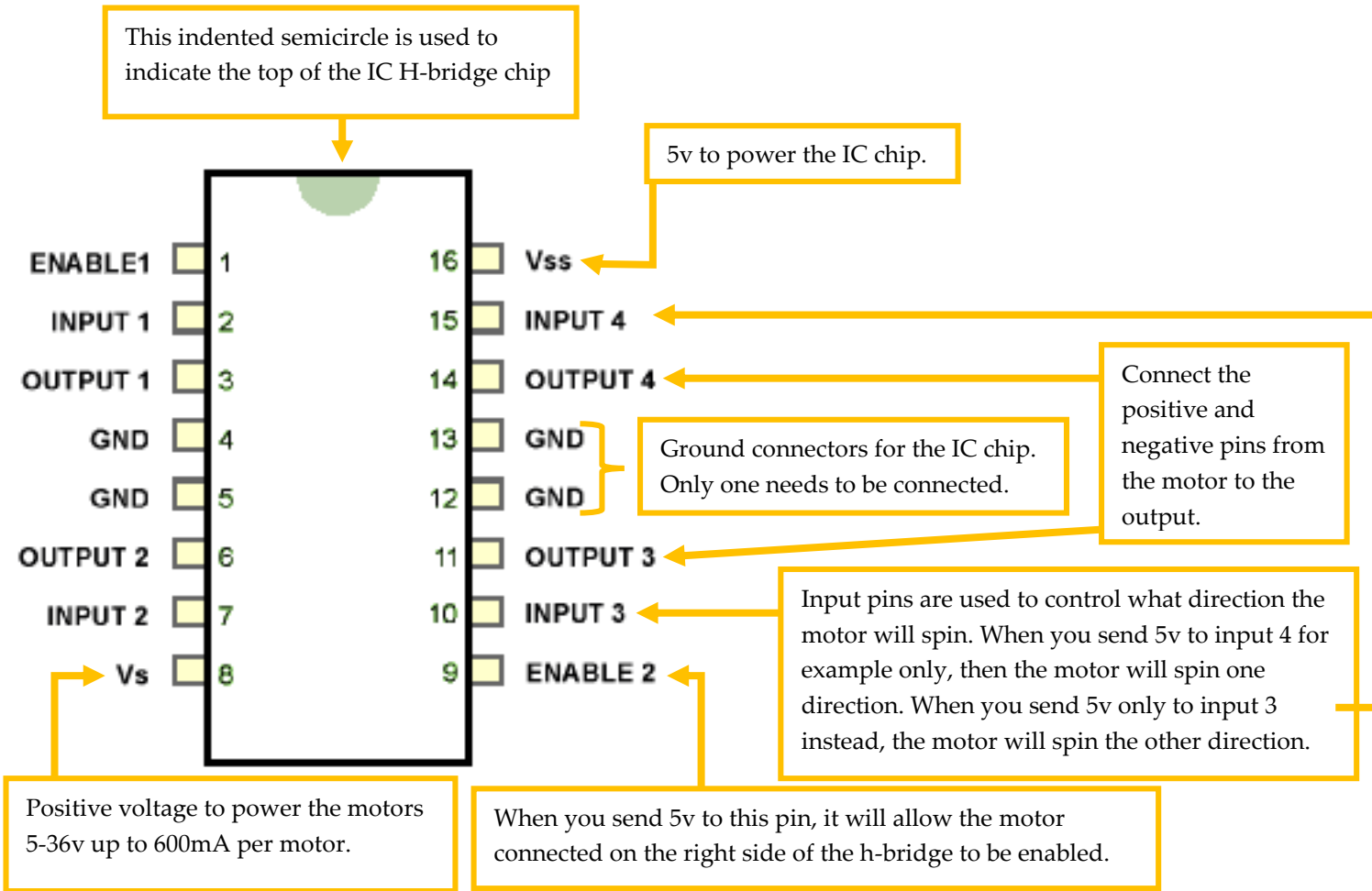
```
Line 29. }
```

What is an H-Bridge?

Time Required: < 15 *minutes*



An H-Bridge is an electric integrated circuit that allows the ability to control up to two motors. These two motors can be independently controlled what direction it will spin. This could allow us to create a robot for example that could move forwards, backwards, turn left and turn right and have our Arduino program those movements. We are just using half the chip for all the projects, most of the pins on the left-hand side of the chip are for controlling a second motor. See the diagram on the next page for what each pin does from the H-Bridge.



H-Bridge Controlled by an Arduino

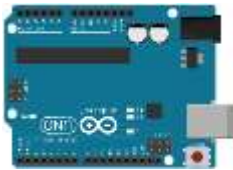
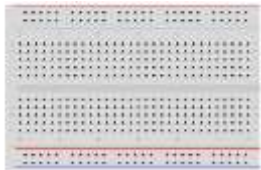



Time Required: 30 minutes

In this project, we will be using an H-Bridge (Model L293D) connected with our Arduino to control the motor so it can rotate clockwise or counter clockwise.

Pin #9 from our Arduino must output HIGH (output 5v) to enable the h-bridge to work. Pins #11 and pin #10 is used to control which direction the motor will turn. If you have pin #11 high (output 5v) the motor will spin one direction, while if you output high on pin #9 it will spin the other direction. The motor will not turn if both pin #10 and pin #11 is high (outputting 5volts) since you are telling the h-bridge to go clockwise and counter clockwise at the same time.

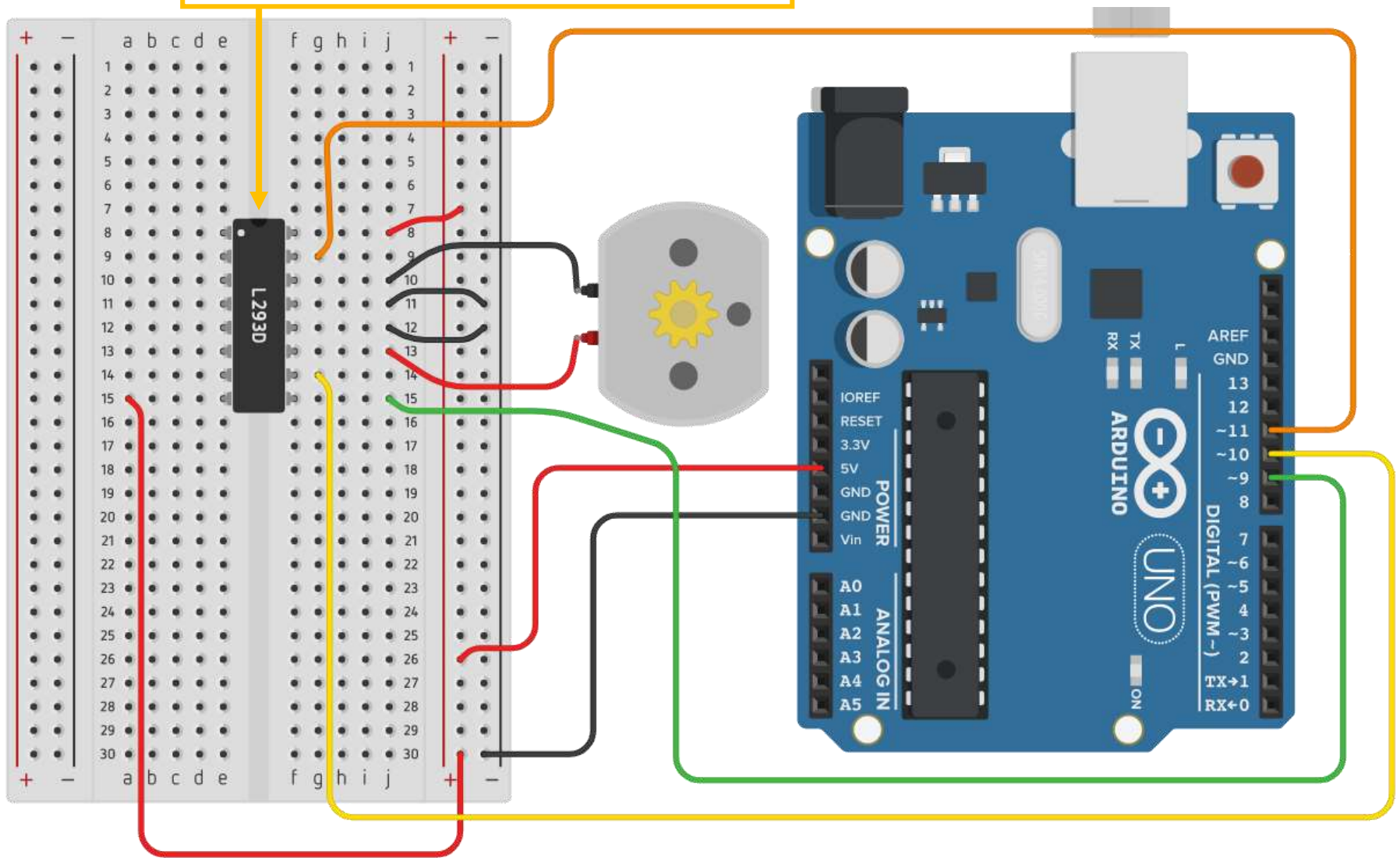
This project will rotate the motor clockwise for 6 seconds, stop spinning the motor for 2 seconds, rotate the motor counterclockwise for 6 seconds, stop the motor for 2 seconds and then repeat forever.

Required Components

				
1x Arduino	1x Breadboard	14x Jumper Wires	1x H-Bridge	1x DC Motor

Wiring

This indented semicircle is used to indicate the top of the IC H-bridge chip. Make sure you put in the h-bridge correctly.



Code

```
Line 1. void setup() {  
Line 2.   pinMode(9, OUTPUT);  
Line 3.   pinMode(10, OUTPUT);  
Line 4.   pinMode(11, OUTPUT);  
Line 5.  
Line 6.   digitalWrite(9, HIGH);  
Line 7. }  
Line 8.  
Line 9. void loop() {  
Line 10.   digitalWrite(10, HIGH);  
Line 11.   digitalWrite(11, LOW);  
Line 12.   delay(6000);  
Line 13.  
Line 14.   digitalWrite(10, LOW);  
Line 15.   digitalWrite(11, LOW);  
Line 16.   delay(2000);  
Line 17.  
Line 18.   digitalWrite(10, LOW);  
Line 19.   digitalWrite(11, HIGH);  
Line 20.   delay(6000);  
Line 21.  
Line 22.   digitalWrite(10, LOW);  
Line 23.   digitalWrite(11, LOW);  
Line 24.   delay(2000);  
Line 25.  
Line 26. }
```

Set pin #9, pin #10 and pin #11 as an output

When the Arduino turns on or restart, set pin #9 as HIGH (on). This pin will be used to tell the h-bridge to be enabled. We did it in the setup since we never need to turn it off later

Set pin #10 to HIGH (on) and pin #11 to LOW (off) and then wait for 6 seconds. This will spin the motor one direction

Set pin #10 to LOW (off) and pin #11 to LOW (off) and then wait for 2 seconds. This will stop the motor from spinning.

Set pin #10 to LOW (off) and pin #11 to HIGH (on) and then wait for 6 seconds. This will spin the motor in the opposite direction.

Set pin #10 to LOW (off) and pin #11 to LOW (off) and then wait for 2 seconds. This will stop the motor from spinning.

H-Bridge Using Potentiometer

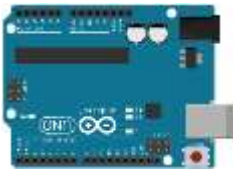
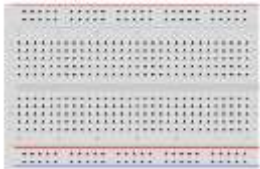




Time Required: 50 minutes

In this project, we will use an H-Bridge (Model L293D) with an Arduino to control the direction of the rotation of the motor based on the position of the potentiometer.

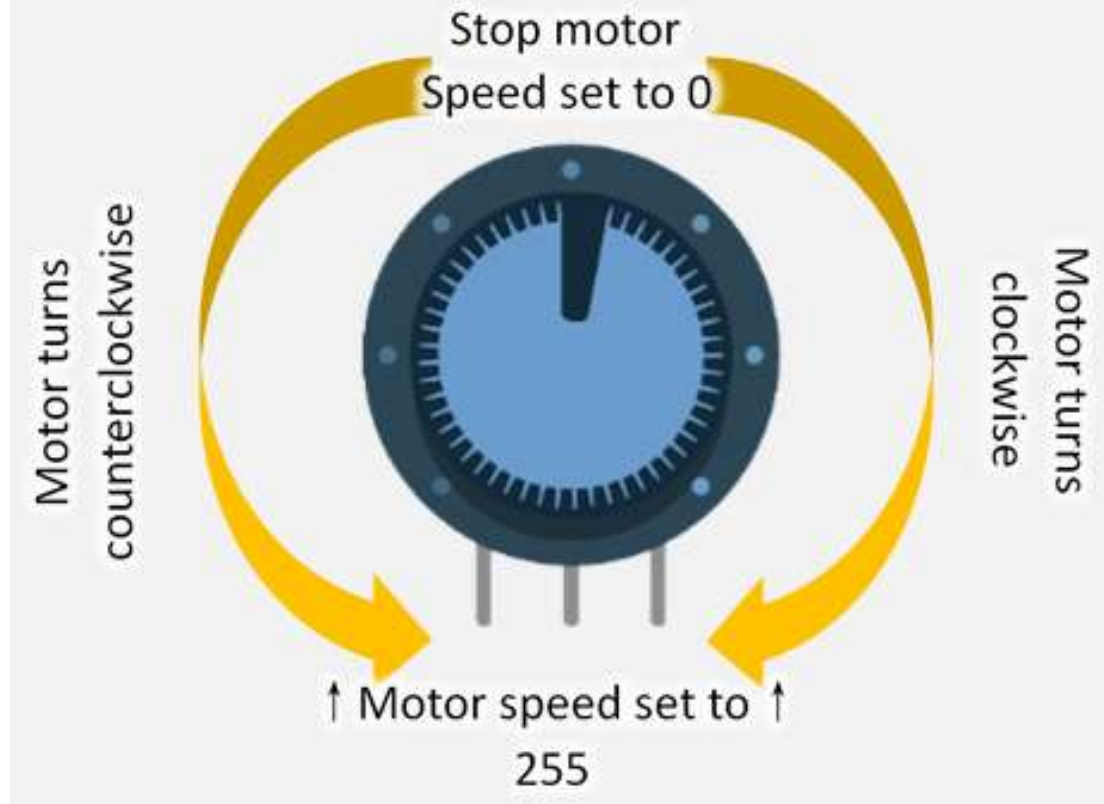
Pin #9 from our Arduino will be used to control the speed of the motor attached (output 5v). Pins #11 and pin #10 is used to control which direction the motor will turn. If you have pin #11 high (output 5v) the motor will spin one direction, while if you output high on pin #10 it will spin the other direction. The motor will not turn if both pin #10 and pin #11 is high (outputting 5v) since you are telling the h-bridge to go clockwise and counter clockwise at the same time.

The potentiometer will be used at the 12 o'clock position to stop all motor movement. If you turn the potentiometer counterclockwise, the motor will spin in that direction as well. If you turn the potentiometer clockwise the motor will spin in the direction as well. As you rotate the potentiometer more and more clockwise or counterclockwise, the motor will increase in speed until you reached the limit of how far it can move in either direction.

Required Components

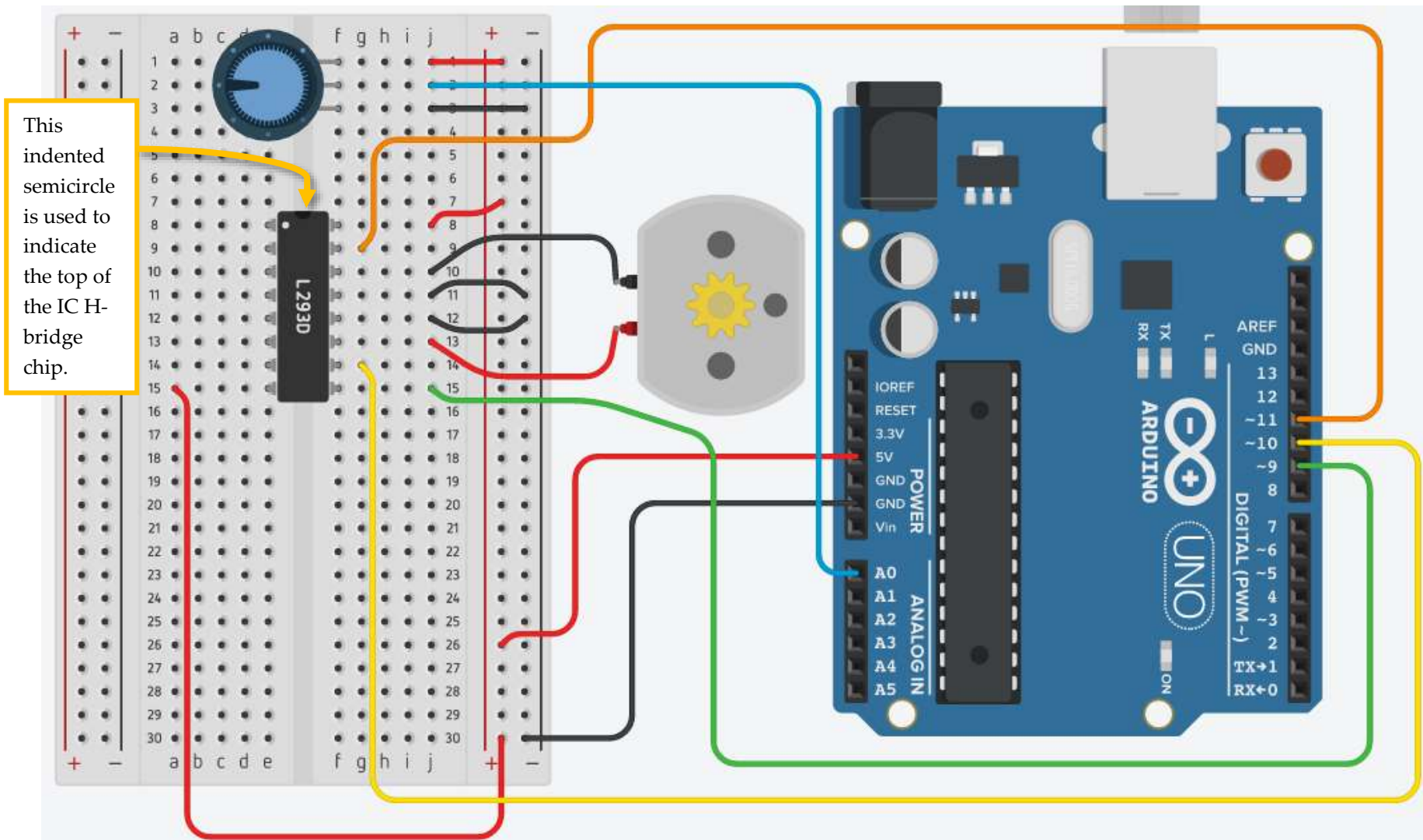
					
1x Arduino	1x Breadboard	12x Jumper Wires	1x Potentiometer	1x H-Bridge	1x DC Motor

Behaviour of Potentiometer for our project



The above diagram above is used to illustrate the movement of the motor when you rotate the potentiometer for this project.

Wiring



Code

```
Line 1.  const int motorEnPin = 9;
Line 2.  const int motorFwdPin = 10;
Line 3.  const int motorBkwdPin = 11;
Line 4.
Line 5.  const int potInputPin = A0;
Line 6.  int inputValue = 0;
Line 7.  int motorSpeed = 0;
Line 8.
Line 9.  void setup() {
Line 10.   pinMode(motorEnPin, OUTPUT);
Line 11.   pinMode(motorFwdPin, OUTPUT);
Line 12.   pinMode(motorBkwdPin, OUTPUT);
Line 13.
Line 14.   pinMode(potInputPin, INPUT);
```

Create a constant integer variable called *motorEnPin* with a value of 9 (the pin to enable the h-bridge to turn on).

Create a constant integer variable called *motorFwdPin* with a value of 10 (the pin is used to tell the h-bridge to move the motor clockwise).

Create a constant integer variable called *motorBkwdPin* with a value of 11 (the pin is used to tell the h-bridge to move the motor counterclockwise).

Create a constant integer variable called *potInputPin* with a value of A0 (the pin is used to read the value from the potentiometer).

Create an integer variable called *inputValue* with a value of 0 (this will be used to store the current value of our potentiometer).

Create an integer variable called *motorSpeed* with a value of 0 (this will be used to store the current value of how fast the motor should spin).

Set our pins for *motorEnPin* (pin #9), *motorFwdPin* (pin #10), *motorBkwdPin* (pin #11) as an *OUTPUT*.

Set the potentiometer that is connected to *potInputPin* (pin #A0) as an *INPUT*.

```

Line 15.   Serial.begin(9600);
Line 16.   }
Line 17.   void loop() {
Line 18.     inputValue = analogRead(potInputPin);
Line 19.
Line 20.     if(inputValue >= 0 & inputValue <= 472) {
Line 21.         Serial.println("Clockwise");
Line 22.         digitalWrite(motorFwdPin, HIGH);
Line 23.         digitalWrite(motorBkwdPin, LOW);
Line 24.         motorSpeed = map(inputValue, 0, 472, 255, 0);
Line 25.     }
Line 26.     else if(inputValue >= 552 & inputValue <= 1023) {
Line 27.         Serial.println("Counterclockwise");
Line 28.         digitalWrite(motorFwdPin, LOW);
Line 29.         digitalWrite(motorBkwdPin, HIGH);
Line 30.         motorSpeed = map(inputValue, 552, 1023, 0, 255);
Line 31.     }

```

Start serial communication when the Arduino boots. Set the communication speed to 9600 baud.

Store the value from the potentiometer using `analogRead` for the `potInputPin` (#A0) into the variable `inputValue`.

When the potentiometer knob input is between 0 – 472, output the Serial message “Clockwise”, set the `motorFwdPin` to `HIGH` and the `motorBkwdPin` to `LOW` (this will make the H-Bridge be told to go clockwise). Store the calculated speed for the motor to spin using the variable `motorSpeed`. We use the `map` function to read the potentiometer so that when the knob is reading 0, the speed will be 255 (speed is a value between 0-255). When the potentiometer is reading 472, set the motor speed to 0. It will then calculate automatically the motor speed to adjust smoothly with the input from the potentiometer.

When the potentiometer knob input is between 552 – 1023, output the Serial message “Counterclockwise”, set the `motorFwdPin` to `LOW` and the `motorBkwdPin` to `HIGH` (this will make the H-Bridge be told to go counterclockwise). Store the calculated speed for the motor to spin using the variable `motorSpeed`. We use the `map` function to read the potentiometer so that when the knob is reading 552, the speed will be 0 (speed is a value between 0-255). When the potentiometer is reading 1023, set the motor speed to 255. It will then calculate automatically the motor speed to adjust smoothly with the input from the potentiometer.

```
Line 32.   else {
Line 33.     Serial.println("Stop");
Line 34.     digitalWrite(motorFwdPin, LOW);
Line 35.     digitalWrite(motorBkwdPin, LOW);
Line 36.     motorSpeed = 0;
Line 37.   }
Line 38.   Serial.print("Input: ");
Line 39.   Serial.println(inputValue);
Line 40.
Line 41.   Serial.print("Speed: ");
Line 42.   Serial.println(motorSpeed);
Line 43.
Line 44.   analogWrite(motorEnPin, motorSpeed);
Line 45.
Line 46.   delay(100);
```

Any other value for the potentiometer knob input (which would be between 473 – 551), output the Serial message "Stop", set the *motorFwdPin* to *LOW* and the *motorBkwdPin* to *LOW* (this will make the H-Bridge be told to stop moving the motor). Set the speed for the motor to 0.

Output the serial message "Input: ", and then output on the same line the contents of the variable from the *inputValue*. Since we are using `println`, this will create a line break after output the contents of that variable.

Output the serial message "Speed: ", and then output on the same line the contents of the variable from the *motorSpeed*. Since we are using `println`, this will create a line break after output the contents of that variable.

Set the speed of the motor. This is done by using `analogWrite` to the *motorEnPin* (pin #9, which is connected to the h-bridge from that pin) outputting the number stored in *motorSpeed*.

Create a delay for 100 microseconds only so we can slow down the serial text output to make it easier to read.

Recommended Resources

Get these for free at the Toronto Public Library

Want to learn more about Arduinos? Here is a list of our favorite Toronto Public Library books and resources. When using the Arduino Kit, please stick to the projects outlined in this manual. Additional projects found in the recommended resources are for educational and entertainment purposes and are only intended for use with your personal Arduino.



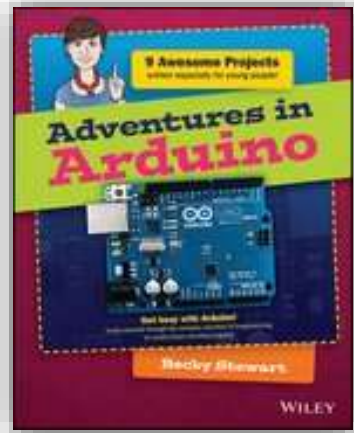
<http://www.Arduino.cc>

The official website has great tutorials and reference resources. The website includes information on all commands you can do for the Arduino programming language and examples on how to use them.



Learning Arduino with Peggy Fisher

This beginner course consists of two hours of video and can be accessed for free from Lynda.com (*via tpl.ca/elearning with a valid Toronto Public Library card*).



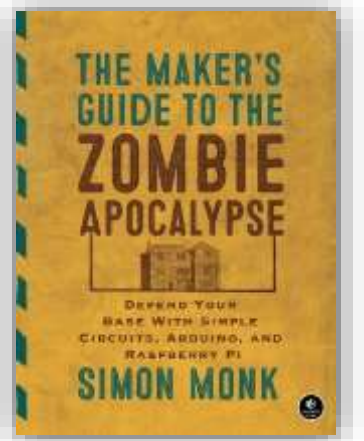
Adventures in Arduino by Becky Stewart

This book provides simple, easy-to-follow introductions to the Arduino. It is written for 11 to 15 year olds, but we've found the concepts, content, and language engaging and applicable to adult Arduino users. *Available from Safari (via tpl.ca/elearning with a valid Toronto Public Library card)*.



Arduino for Kids (2017) by Priya Kuber, Rishi Gaurav Bhatnagar, Vijay Varada

This book is intended for children (ages 9 and up) and their parents. It includes a series of fun, easy projects that don't require any knowledge of electronics. *Available from Safari (via tpl.ca/elearning with a valid Toronto Public Library card).*



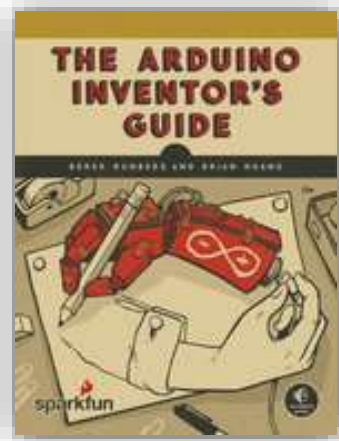
The Maker's Guide to the Zombie Apocalypse: Defend Your Base with Simple Circuits, Arduino, and Raspberry Pi (2016) by Simon Monk

No one knows what the future holds, so we can't definitively say whether or not the Arduino projects in this book will come in handy. What we can guarantee is that you'll have fun learning about Arduinos in a unique and creative way. *Available in regular print.*



Make: Drones: Teach an Arduino to Fly by David McGriffy

Have you ever wondered how drones work? This book reveals drone building secrets and explains how you can get your Arduino to fly. *Available in regular print and as an eBook from Safari (via tpl.ca/elearning with a valid Toronto Public Library card).*



The Arduino Inventor's Guide (2017) by Derek Runberg and Brian Huang

Ready to move on from the Arduino kits and start working on some more advanced projects? Why not build a tiny electric piano, a desktop greenhouse, or a colour-mixing night light? You'll find ten fun Arduino projects in this new eBook, *available from Safari (via tpl.ca/elearning with a valid Toronto Public Library card)*



Arduino Playground : Geeky Projects for the Experienced Maker (2017) by Warren Andrews

This is the perfect resource for more advanced Arduino projects. One of our favourites is the *Garage Sentry Parking Assistant*, a project that can help you pull into your garage by setting off an alarm when you've gone far enough and need to hit the brakes. *Available in regular print and as an eBook from Safari (via tpl.ca/elearning with a valid Toronto Public Library card)*